Juan Barrera
Professor Jeffrey Seaman
Survey Of Programming Languages
Summary Module 3

This week, I learned about how to develop web applications using Spring Boot, which is a robust framework for constructing Java based web apps. The main emphasis was on developing a RESTful API and recognizing how Spring Boot streamlines the development process through automatic management of dependencies, configuration, and setup.

So first I started by establishing the Spring Boot project through Spring Initializr, where I discovered how to build a Spring Boot application by choosing dependencies such as Spring Web and Spring Boot DevTools. These dependencies enable me to easly generate web endpoints and automatically refresh the application when modifications occur.

One of the first things I had to do was configuring the Controller class. In Spring Boot, @RestController is utilized to create a controller responsible for managing HTTP requests. I set up fundamental REST endpoints, beginning with a straightforward GET request to send a welcome message to the user. This section helped me to understand better how the Spring framework associates HTTP requests to Java methods by utilizing annotations such as @GetMapping and @RequestParam for managing dynamic data.

I encountered several problems concerning the directory arrangement and environment configuration. Initially, there was uncertainty regarding the project folder and its organization, because I didn’t know I had to set up the environment in a certain way (folder structure). Because of this, the package within the Java file did not match the folder structure, leading to errors during application execution. After I fixed the directory and package configuration, I could compile and execute the project. A crucial lesson was the significance of making sure the package

declarations correspond to the directory layout for Java applications to function correctly in Spring Boot.

An important thing I learned was managing dependencies. In Spring Boot, dependencies are handled via Maven utilizing the pom.xml file. I included dependencies such as Spring Boot starter libraries for both web and logging. Executing the mvn clean install command lets me compile the project and address any absent dependencies.

While developing, I faced several problems, including 404 errors when trying to access specific endpoints in the browser. I dedicated some time resolving these errors by examining the mappings in the Controller class and ensuring the endpoints were properly defined. I needed to comprehend the Whitelabel Error Page that Spring Boot displays when an endpoint cannot be located. After troubleshooting the problem, I effectively mapped the endpoints and was able to access them in the browser.

I also understood the significance of adhering to the Spring Boot convention rather than configuration method. The Spring framework provides reasonable defaults, allowing me to depend on Spring Boot to manage the majority of the setup instead of manually configuring every minor detail of the application. This significantly decreased the amount of boilerplate code I needed to create and accelerated development.

Ultimately, I dedicated time to grasping the complete project framework. In a Spring Boot application, files are usually structured in this manner:

src/main/java: Houses the source code, including controllers, services, and models.

src/main/resources: Contains configuration files and static resources (such as HTML or CSS files, in the case that they are needed).

pom.xml: This is the configuration file for Maven, managing dependencies, plugins, and various settings related to the project.

This week, I additionally discovered how to incorporate error handling in a Spring Boot application. By utilizing @RestControllerAdvice and @ExceptionHandler, I addressed exceptions like absent parameters or erroneous data formats. This functionality was crucial for ensuring the API's strength and ease of use by delivering suitable error messages and status updates